B2000++ Lecture Notes
Using B2000++

Version1.2 of 13.1.2014

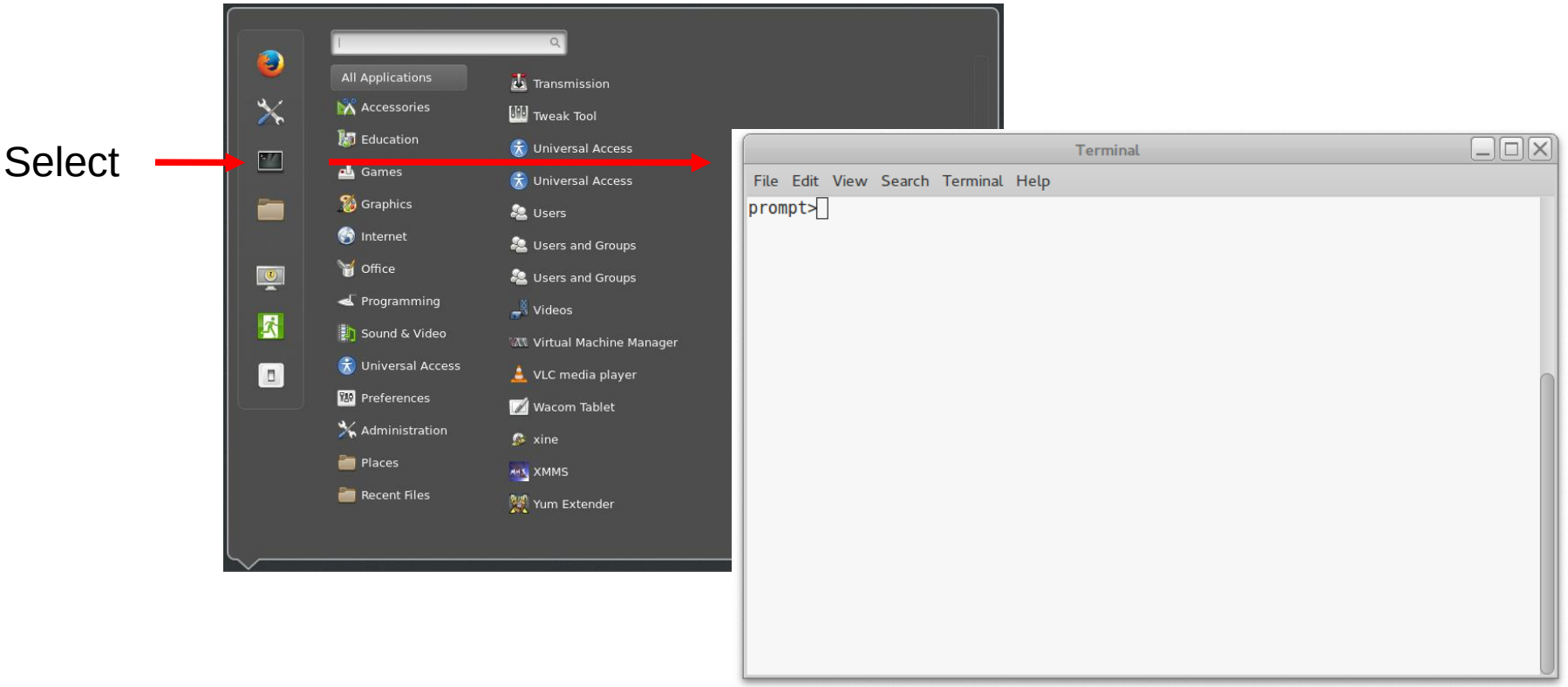
SMR Engineering & Development
CH-2500 Bienne
http://www.smr.ch

Content of Lecture

- Presentation of an example – Buckling of a plate
- Create a FE model with B2000++ with the Model Description language (MDL) or with the Nastran BDF converter
  - Overview of MDL
  - Summary of MDL model creation capabilities
- Execute B2000++
- Summary of post-processing capabilities
  - Viewing
  - Scripting

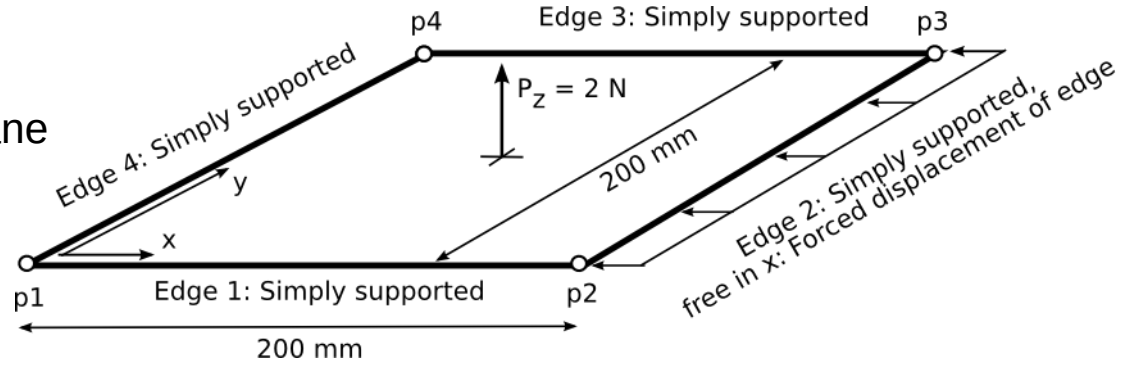
B2000++ and the Unix shell

- Meant for those who are not familiar with Unix terminals and the Unix shell.
- B2000++ and the tools must be launched from a terminal.
- To this end, you must create a terminal with the Gnome or Cinnamon Desktop. Example (Cinnamon desktop):



Example: Post-buckling of Plate

- Example to illustrate the non-linear solid mechanics analysis capabilities of B2000++.
- Square plate made of laminated composite material.
- Loaded in-plane by a “displacement load” and out-of-plane by a concentrated load.
- Plate is simply supported.
- B2000++ shall compute:
  - The linear buckling load due to in-plane “displacement loading”
  - The non-linear response due to increasing in-plane “displacement loading”



- Note: The example can be found in the examples collection under

```
<prefix>/share/doc/B2000++/examples/solid_mechanics/
static_plate_postbuckling
```

Copy the examples to one of your directories:

```
cp -a <prefix>/share/B2000++/test_cases/examples .
```

Example: Post-buckling of Plate

Buckling analysis: Go to directory

`examples/solid_mechanics/static_plate_postbuckling/buckling`

and run B2000++ and the buckling value extraction post-processing script by typing

`make`

Output from B2000++ Solver

```
INFO:solver.linearised_prebuckling:12:26:54.108: Start linearised prebuckling solver for case 1.
INFO:solver.linearised_prebuckling:12:26:54.108: Assemble the linear problem.
INFO:solver.linearised_prebuckling:12:26:54.111: Element matrix assembly
INFO:solver.linearised_prebuckling:12:26:54.395: Resolve the linear problem
INFO:solver.linearised_prebuckling:12:26:54.447: Assemble the stability matrix.
INFO:solver.linearised_prebuckling:12:26:54.477: Eigenvalue problem resolution
INFO:solver.linearised_prebuckling:12:26:54.802: End of linearised prebuckling solver
```

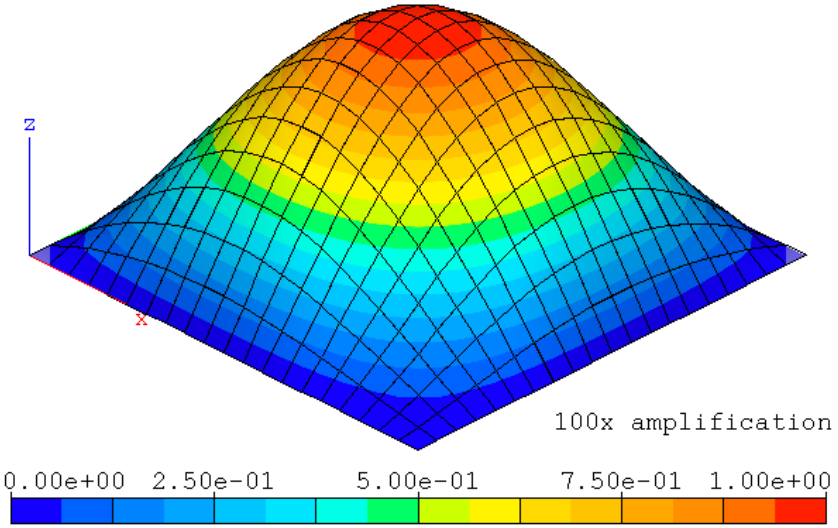
Output from Python post-processing script

```
Total force Fx [N]= 89373.4838994

Eigenvalue= 0.039875218639

Fcrit=Fx*Eigenvalue [N]= 3563.78721101

Ncrit=Fcrit/L [N/mm]= 17.8189360551
```



Post-processor (`make view`): Buckling shape and amplitude

Example: Post-buckling of Plate

Nonlinear analysis: Run a B2000++ non-linear analysis in directory

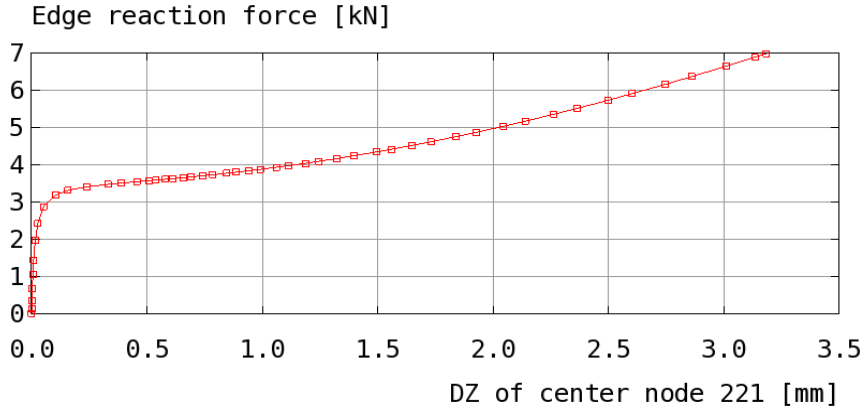
```
examples/solid_mechanics/static_plate_postbuckling/post_buckling
```

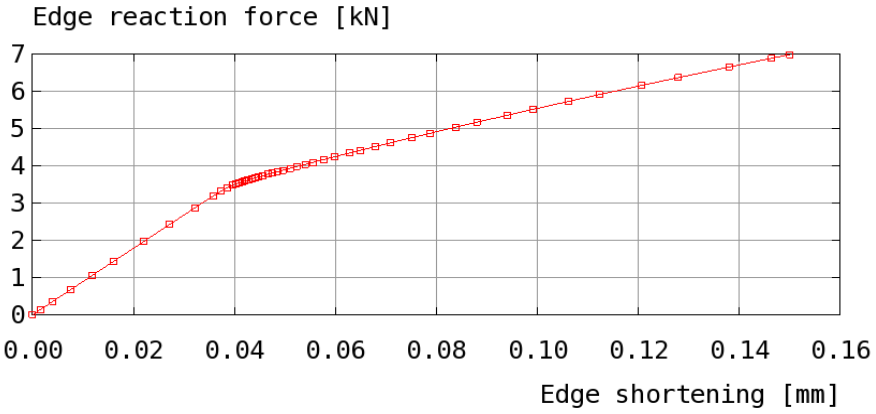
by typing

```
make
```

- Plot the results with a script for plotting the total edge reaction force vs. plate center out-of-plane displacement (upper diagram) and total edge reaction force vs. edge shortening (lower diagram):

```
make view
```





Example: Run the Analysis

The example plate problem described by the MDL text input file **`shell.mdl`** can be executed in 2 modes:

1. Launch the solver with the MDL input file:

   ```
   b2000++ shell.mdl
   ```

   creating the model database **`shell.b2m`** and solving the problem.

2. Run the input processor, followed by the B2000++ solver:

   ```
   b2ip++ -i shell.mdl
   ```

   creating the model database **`shell.b2m`**. Solve with

   ```
   b2000++ shell
   ```

Example: Run the Analysis

- To get information or debug information during the execution of the solver, issue one of the solver options. Note that

  `b2000++ -h`

  will print a short list of options and stop.

- The **`-l`** option of b2000++ allows for printing all kind of information.

  `b2000++ -l 'info of solver in cerr' shell.mdl`

  will print a short summary of the solver steps. Plate example:

```
INFO:solver.linearised_prebuckling:12:26:54.108: Start prebuckling solver for
case 1.
INFO:solver.linearised_prebuckling:12:26:54.108: Assemble  linear problem.
INFO:solver.linearised_prebuckling:12:26:54.111: Element matrix assembly.
INFO:solver.linearised_prebuckling:12:26:54.395: Resolve linear problem
INFO:solver.linearised_prebuckling:12:26:54.447: Assemble stability matrix.
INFO:solver.linearised_prebuckling:12:26:54.477: Eigenvalue problem resolution.
INFO:solver.linearised_prebuckling:12:26:54.802: End of prebuckling solver.
```

Model and Analysis

- An analysis with B2000++ is performed in two steps:
  1. Create the B2000++ model database,
  2. Solve the problem.

  Both steps are executed in batch mode.
- The solver is capable of creating the database before solving the problem. The is is the usual procedure, unless one wants to view the model before starting the solver.
- A B2000++ model database can be created by
  - Specifying the mesh, boundary conditions and solution parameter with the B2000++ Model Description Language (MDL),
  - Converting a Nastran BDF file to an (almost) complete MDL file,
  - Creating a mesh with Salome, convert it to MDL, and add boundary conditions and solution parameter with the B2000++ Model Description Language (MDL),
  - User-written software,

  and by translating the MDL file to a B2000++ database with the B2000++ input processor b2ip++.

Generate Model with MDL Syntax

- A B2000++ model is generated by means of a text file containing MDL commands. The file is passed to the B2000++ input processor which translates it to database format.
- The B2000++ Model Description language is a relatively simple, easy to understand keyword-based language with some flow control capabilities. Example:

```
set 1
   title 'Forced displacement'
   value -1.0 dof 1 epatch 1 e2
end
```

  which means *create a set no. 1 with a title and with values of -1.0 for dof (degrees-of-freedom) 1 assigned to patch 1, edge e2*.

- For details, see B2000++ user Manual:

  B2000++ User Manual > The B2000++ Model Description Language (MDL) > MDL syntax and programming

Generate Model with MDL Input Deck

- A B2000++ Model Description Language (MDL) input deck is composed of a series of blocks, each of them defining a specific feature of the model.
- Each block is started by the relevant keyword and terminated by end.
- Example node coordinate definitions

```
nodes
   1  0.0 0.0 0.0
   2  2.5 0.0 0.0
   dref 1
   3  5.0 0.0 0.0
   dref 0
   4  0. 5. 0.0
end
```

Nodes 1 and 2 adopt default DOF reference system, node 3 adopts DOF reference system 1, and node 4 and subsequent default system 0.

Generate Model with MDL Input Deck

```
# Title
title " "

# Branches definitions
branch id
  commands
end

# Element materials definitions
emat
  mid id
    commands
  endmid
end

# Explicitly connect nodes
join
  commands
end

# Linear constraints
linc
  commands
end
```

```
# Essential boundary conditions
ebc
  commands
end

# Natural boundary conditions
nbc
  commands
end

# Analysis cases
cases
  commands
end

# Analysis directives
adir
  commands
end
```

Generate Model with MDL Branch Definition

- B2000++ allows for defining branches, i.e. parts of a model with independent node, element, and boundary condition definitions. Note: If only one branch exists in the model, the branch does not have to be defined explicitly.
- Branches can then be connected with the join command.
- NASTRAN does not have the concept of branches. Thus, for NASTRAN models there is only one branch, branch no. 1.
- In case there is only one branch the branch block does not have to be defined.
- Branch block:

```
branch id
   nodes
      ...
   end
   elements
      ...
   end
   ...
end
```

```
branch id
   epatch id
      ...
   end
   ...
end
```

Generate Model with MDL: Nodes

- The **nodes** block:

```
nodes
   id x y z
   ...
   dref id
   cref id
   ...
end
```

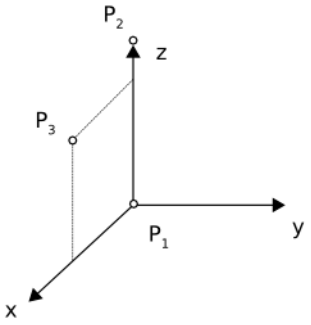
- `id` designates the node identifier (a positive integer), the 'external node identifier'.
- `x, y, z` designate the coordinates with respect to the current transformation.
- `dref` designates the DOF transformation. Default id 0 is branch system.

Generate Model with MDL: Transformations

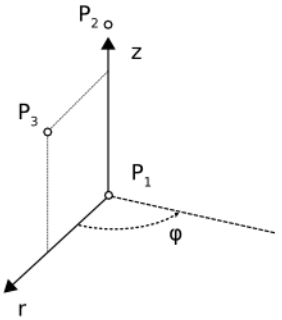
- The **transformation** block:

```
transformation
   id type p1x p1y p1z p2x p2y p2z p3x p3y p3z
   ...
end
```

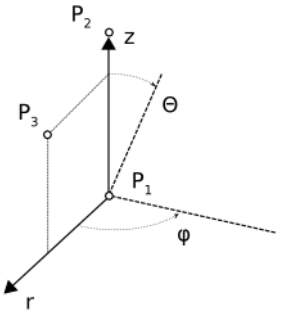
- `id` designates the transformation identifier (a positive integer).
- `type` designate the type of transformation (`CARTESIAN`, `CYLINDRICAL`, `SHPERICAL`).
- The three points p_1, p_2, and p_3, construct the transformation.



Cartesian



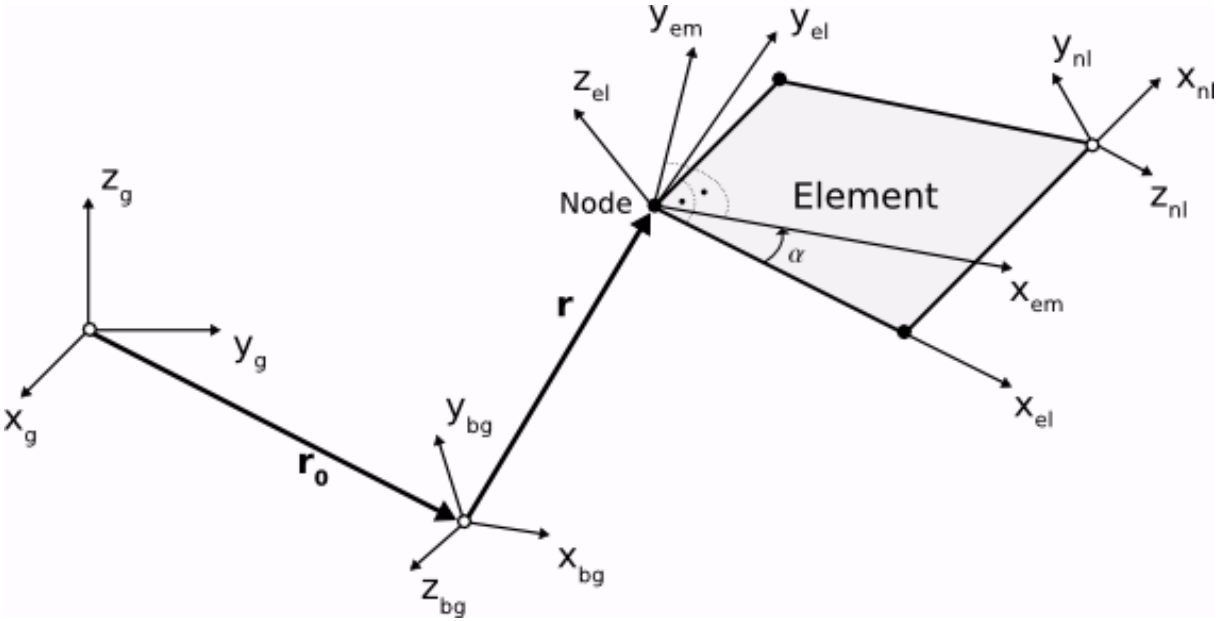
Cylindrical



Spherical

Generate Model with MDL: Reference Frames

- B2000++ discerns between the following coordinate systems reference frames).
  - Global system
  - Branch system
  - Element system
  - Material system.



- Usually and per default the global system and the branch system coincide, as is the case with NASTAN meshes.

Generate Model with MDL: Elements

- B2000++ elements can be specified one by one, as is the case when NASTRAN BDF meshes are translated.
  - Requires the nodes to be defined.
  - allows for maximum flexibility with definition of material properties, shell properties, local element orientation, etc.
- B2000++ elements can be generated with the **`epatch`** command – a means for generating meshes based on simple analytical geometries.
  - A patch defines nodes and elements.
  - Node and element attributes are constant throughout the patch.

Generate Model with MDL: Elements

- The elements block:

```
elements
  attributes ...
  eid n1 n2 ... nnne
  attributes ...
  eid [n1 n2 ... nnne]
end
```

- Required attributes are the element type **`eltype t`** which also re-initializes all attribute definitions within the block and, for most element types, the material identifier **`mid id`**.
- Other attributes are element dependent, see element documentation.
- The element identifier **`eid`**, together with the element-to-node connectivity

  $\mathbf{n_1\ n_2\ \ldots\ n_{nn}}$

  defines elements with the attributes previously listed (or the default attributes). Brackets are required in case the number of nodes is variable.
- See the *Generic Element* section of the user manual for the element node, edge, and face numbering scheme.

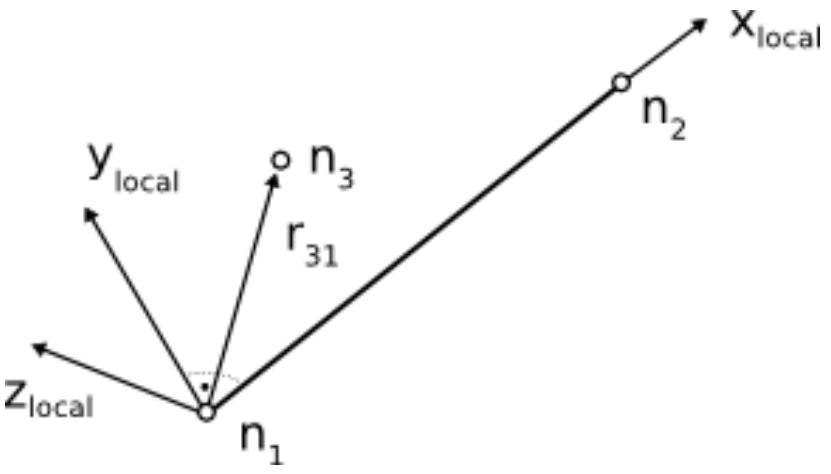
Generate Model with MDL: Elements

SMR Engineering & Development

- Example of element definition:

```
elements
   eltype B2.S.MITC
   mid 1
   section rectangular 0.3 0.01
   beam_orientation 0 1 0
   1 1 2
   2 2 3
   ...
end
```

which defines 2 node beam elements. Note that beam elements require the definition of a local frame. This is achieved with the **`beam_orientation`** defining the local beam y-axis.



Generate Model with MDL: Element patches

SMR Engineering & Development

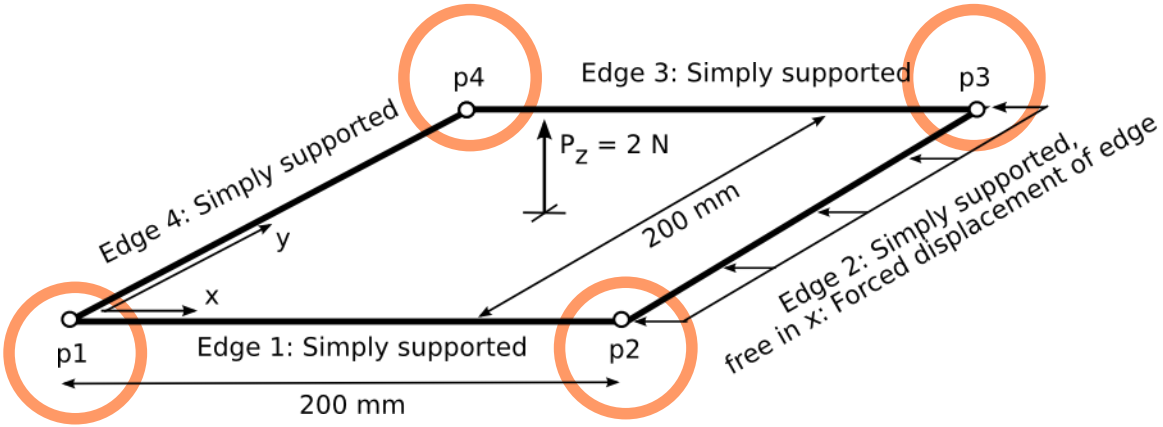
- Elements can be generated with patches, i.e. some simple pre-defined analytical geometrical forms such as 4-sided plate, cylinder, cone, cube, etc.
- Element patch block definition:

```
epatch id
   geometry t
   geometrical_attributes ...
   eltype t
   element_attributes ...
end
```

- One or more element patches can be generated in one and the same branch.
- However, some branches usually require translation and rotation. In this case only one patch per branch!
- More than one element patch require the **`join`** command (see later).

Generate Model with MDL: Element patches

- From the Introduction Example: Generate the mesh and the elements of the plate with a MDL epatch (here: with id=1).




```
epatch 1
   geometry plate
   p1   0.   0.   0.
   p2 200.   0.   0.
   p3 200. 200.   0.
   p4   0. 200.   0.
   ne1 20 ne2 20     # N. of elements in either direction
   eltype Q4.S.MITC  # Element type generated
   thickness 1.6     # Thickness (for non laminate material)
   mid 2             # Material identifier
 end
```

Generate Model with MDL: Material

- The `emat` block defines all element materials which describe the physical properties of the materials assigned to the elements.

```
emat
  mid id
    type t
    attributes
  end
  ...
end
```

- Each material is identified by a positive integer **`id`** and a material type descriptor **`t`**.
- The element material is then referenced from elements etc. by the identifier **`id`**.

Generate Model with MDL: Material

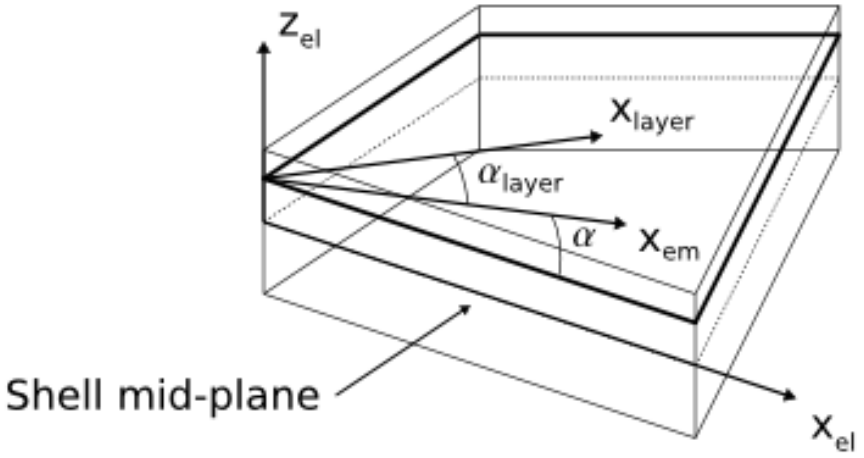
- Example from Introduction:
  - Define an orthotropic stress analysis material **1** for each layer of MITC shell elements. Define a second material of type LAMINATE.
  - The shell element then gets the material reference **2** (the LAMINATE material), which in its turn references the layer material.

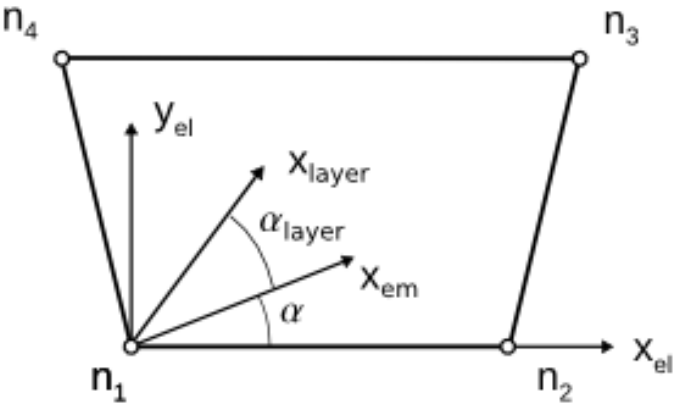
```
emat
   mid  1
      type ORTHOTROPIC
      e1 146.86e+03 e2 9.65e+03 e3 9.65e+03
      p1 0.3 p2 0.023 p3 0.023
      g1 45.5e+02 g2 45.5e+02 g3 45.5e+02
   end
   mid 2
      type LAMINATE  lid 1
   end
```

Generate Model with MDL: Laminates

- The **`laminates`** block allow for defining all laminates of the model, the laminates being placed in a sub-block identified by a positive identifier **`id`**.
- Each ply contains the ply number **`iply`**, the ply thickness **`t`**, the angle of orientation **`alpha`**, and the material identifier **`mid`**.

```
laminates
  lid id
    [attributes]
    iply t alpha mid
    ...
    [attributes]
    iply t alpha mid
    ...
  end
  ...
end
```





Laminate orientation in element

Generate Model with MDL: Essential B. C.

SMR Engineering & Development

- The **ebc** block defines essential boundary conditions, such as 'zero constraints' or 'displacement constraints'.
- Collections of conditions are grouped in a set identified by a positive integer. Later, in the **cases** blocks, these sets are referenced.

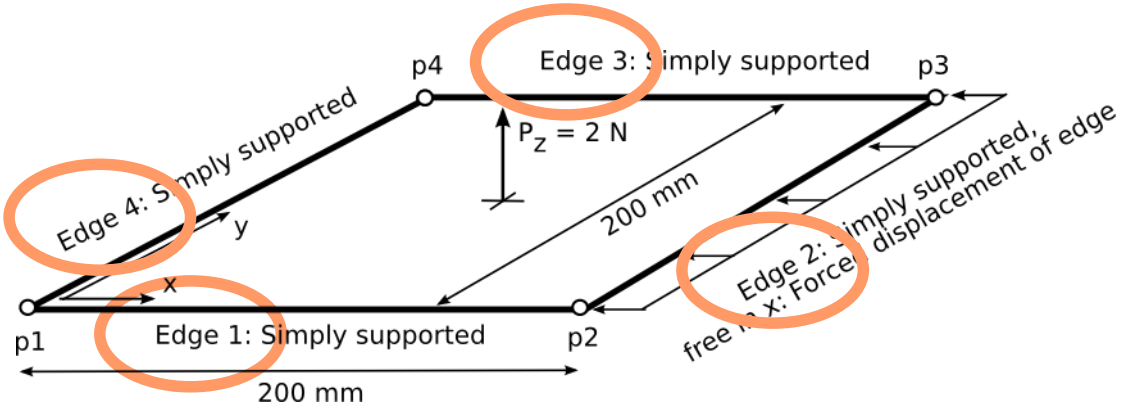
```
ebc
  set id
    attributes
    list ...
    attributes
    list ...
  end
  ...
end
```

- Conditions can be generated for nodes, edges, faces, volumes and they can be assigned to edges or faces of patches.
- Again, attributes are listed, followed by the objects such as nodes, edges, etc., to which they will be assigned.
- See B2000++ User Manual

  The B2000++ Model Description Language (MDL) > Global MDL Commands > ebc

Generate Model with MDL: Essential B. C.

- Example from Introduction:
  - Create a *set 1* with non-zero constraints for dof 2 of edge 2 of epatch 1. This is the edge which is going to be pushed.
  - Create a *set 2* with zero constraints for DOF's 3 of edges 2 and 3 of epatch 1 and DOF's 1 and 3 of edge 4. Add an additional constraint to node 1 for dof 2.
    This is the set which defines the plate support conditions




```
ebc
   set 1
      title 'Forced displacement'
      value -1.0 dof 1 epatch 1 e2
   end
   set 2
      title 'Zero constraints'
      value 0.0 dof [3] epatch 1 e1
      value 0.0 dof [3] epatch 1 e2
      value 0.0 dof [3] epatch 1 e3
      value 0.0 dof [1 3] epatch 1 e4
      value 0.0 dof 2 nodes 1
   end
end
```

Generate Model with MDL: Natural B. C.

- The `nbc` block defines natural boundary conditions, such as 'forces' or 'heat sources'.
- Collections of conditions are grouped in a set identified by a positive integer. Later, in the `cases` blocks, these sets are referenced.
- Conditions can be generated for nodes, edges, faces, volumes and they can be assigned to edges or faces of patches.

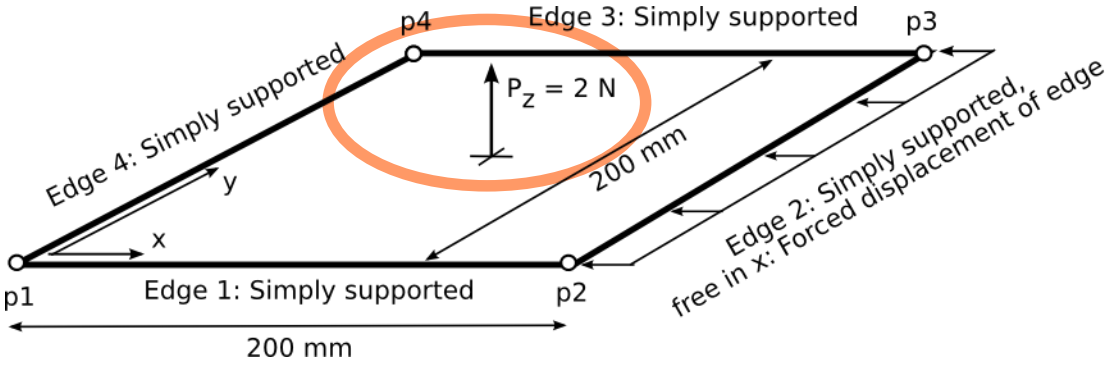
```
nbc
  set id [parameters]
    attributes
    list ...
    attributes
    list ...
  end
  ...
end
```

- Again, attributes are listed, followed by the objects such as nodes, edges, etc., to which they will be assigned.
- See B2000++ User Manual

  The B2000++ Model Description Language (MDL) > Global MDL Commands > nbc

Generate Model with MDL: Natural B. C.

- Example from Introduction:
  - For the one epatch model: Create the *set 2* with a force of 2 N in the z-direction (DOF 3) of node 221. This node can be parametrized with the MDL scripting capability and the definition of variables (see input file **`plate1.mdl`**).
  - For the four epatch model: Create the *set 2* with a force of 2 N in the z-direction (DOF 3) of point **`P3`** of **`epatch 1`**. Node is parametrized!




```
# One-branch model
nbc
   set 2
      title 'Out-of-plane force'
      value 2. dof 3 nodes 221
   end
end
```

```
# Four-branch model
nbc
   set 2
      title 'Out-of-plane force'
      value 2. epatch 1 p3
   end
end
```

Generate Model with MDL: Case definition

SMR Engineering & Development

- The **`cases`** block defines all ingredients needed for one or more specific states referred to as cases. One or more cases can be defined in the **`case`** sub-blocks. Each case is identified by the case identifier **`id`**.

  Specifically, a case

  - Adds boundary conditions sets to a case
  - Defines set names (optional)
  - Defines solution type and strategy

```
cases
  case id
    attributes
    end
  case id
    attributes
  end
  ...
end
```

- Many of the attributes in a **`case`** are solution type dependent. The relevant sections of the user manual explain all parameters pertaining to a specific solution type.

Generate Model with MDL: Case definition

SMR Engineering & Development

- Examples from Introduction: The first analysis calculates the linear buckling load under the given boundary condition configuration. The associated case definition is

```
cases
   case 1
     analysis linearised_prebuckling  # Analysis type
     nmodes 10          # Compute 10 eigenmodes around 0.0
     ebc 1              # Include displacement constraint set 1 (ebc)
     ebc 2              # Include zero-value constraint set 2 (ebc)
     nbc 2              # Include Force set 1 (nbc)
   endcase
end
```

- Note the **`analysis`** keyword hierarchy:
  - The **`analysis`** method analysis specified in the **`adir`** command defines the overall analysis method for all cases, except when **`analysis`** is specified in the **`case`** block, where is supersedes the overall one.

Generate Model with MDL: Case definition

SMR Engineering & Development

- Examples from Introduction: The second analysis calculates the nonlinear response under the given boundary condition configuration. The associated case definition is

```
cases
   case 1
      analysis nonlinear
      increment_control_type load # Load control (no continuation)
      ebc 1             # Include displacement constraint set 1 (ebc)
      ebc 2             # Include zero-value constraint set 2 (ebc)
      nbc 2             # Include Force set 1 (nbc)
      step_size_init 0.01 # Initial step size
      step_size_max 0.1   # Maximum step size
                          # to get some points on the curve!
      step_size_min 0.001 # Minimum step size (IMPORTANT!)
   endcase
end
```

Generate Model with MDL: Analysis Directives

SMR Engineering & Development

- The **adir** tells the solver which cases to process during the run. ***list*** contains one or more `case` identifiers. If more than one identifier is specified, the identifiers must be placed between square brackets [...].

```
adir
  cases list
  Other attributes…
end
```

- Example from the Introduction:

```
adir
  cases 1
end
```

```
adir
  cases [1]
end
```

```
adir
  cases [1 2]
end
```

- See B2000++ User Manual

The B2000++ Model Description Language (MDL) > Global MDL Commands > cases

Generate Model with MDL Other Commands

- **`title`** Specifies an optional problem title.
- **`join`** connects branches and/or patches.
- **`linc`** defines linear constraints.
- **`dof_init`** and **`dofd_init`** define initial conditions for transient analysis.

Generate Model with MDL: Example database

- The database can be browsed with the **b2mcbrowser** data browser or with **baspl++**
- Example database directory:

Contents of database file "./shell.b2m/archives.mc":

dataset_name	type	file_address	size	dsize	ndim	dimensions
ADIR	I	4615240	1	8192	1	1
BDTB.1	$	4263072	8192	0	1	8192
BMODE.1.0.0.1.1	F	4918728	2205	4096	2	441, 5
...						
BMODE.1.0.0.1.10	F	5266416	2205	4096	2	441, 5
BNFP.1	I	4525856	2205	4096	2	441, 5
CASE.1	$	4598848	12288	4096	2	3, 4096
CONSTANTS	$	4197504	4096	0	1	4096
COOR.1	F	4538776	1323	0	2	441, 3
DISP.1.0.0.1	F	4841464	2205	4096	2	441, 5
EBC.1.0.0.1	F	4561216	63	4096	2	21, 3
EBC.1.0.0.2	F	4565816	318	4096	2	106, 3
ELEMENT-PARAMETERS	$	0	4096000	8192	2	1000, 4096
ELMA.1	I	4556416	1200	0	2	400, 3
EMAT.0.0.0.1	F	4581320	68	4096	1	68
…						
ETAB.1	$	4275360	240000	4096	2	400, 600
FIELDS	$	4201600	61440	0	2	30, 2048
FORC.1.0.0.1	F	4802832	2205	4096	2	441, 5
LAMI.0.0.0.1	F	4576576	80	4096	2	16, 5
NBC.1.0.0.2	F	4572456	3	4096	2	1, 3
NODA.1	I	4549360	1764	0	2	441, 4
NODE-NORMALS.1	F	4775352	1323	0	2	441, 3
NODE-PARAMETERS	$	4104192	9216	0	2	18, 512
NODS.1.0.270.0	I	4519456	1600	0	2	400, 4
PATCH.1	$	4271264	4096	0	1	4096
RCFO.1.0.0.1	F	4880096	2205	4096	2	441, 5
SOLUTION.0.0.0.1	$	5288152	4096	0	1	4096

Convert Nastran BDF deck

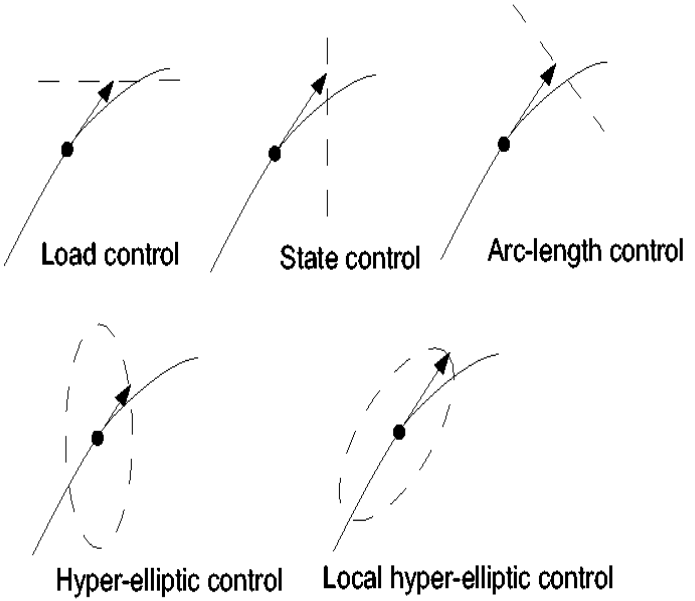
- The Nastran BDF converter reads a Nastran BDF file and translates (most) of the deck to a B2000++ MDL file.

```
b2convert_from_nas [options] bdf-file mdl-file
```

- Options:
  - `-coor-scale=factor` scales all coordinates, thicknesses, and offsets by a factor. Also scaled are non-structural masses.
  - `-dof-scale=factor` scales all specified DOF's (ebc's) by a factor.
  - `-force-scale=factor` scales all specified forces (nbc's) by a factor.
  - `-permute=[X|x|Y|y|Z|z]` specifies permutation for the x-, y-, and z-axes.
  - `-verbose` produces verbose output.
- Note that BDF instructions not understood by the converter are listed in the beginning of the generated MDL file.

B2000++ Solvers: Some General Remarks

- Each type of solution is defined by the `analysis` keyword in the cases block.
- The *Solvers* section of the user manual explains in detail each set of parameters pertaining to a specific analysis type.
- The non-linear solver will be explained here in more detail. The `increment_control_type` parameter of `case` specifies the predictor type `t`:
  - `load` specifies load control
  - `state` specifies state control
  - `hyperplane` specifies hyperplane control (default)
  - `hyperelliptic` specifies elliptic hyperplane control
  - `local_hyperelliptic` specifies local elliptic hyperplane control



Post-Processing: The baspl++ Viewer

- baspl++ is a viewing and post-processing tool fully integrated in B2000++:
  - It allows for viewing most of the mesh and solution of a B2000++ model.
  - It also allows for on-line viewing (during solution process).
  - The Python based scripting language allows for extracting and displaying of text-based output.
- In addition, baspl++ also understands other solvers and formats for multi-disciplinary analysis (not discussed here).
- This short introduction describes
  - Some elementary viewing functions
  - Some elementary scripting functions

The baspl++ Viewer: Launch

- baspl++ is launched from the terminal (shell) with

  baspl++ ***[options] [database(s)] [script [argument(s)]***

  or from a Python script (see later) or with the GUI.

- Example: Launch baspl++ for viewing the plate problem of the introduction

  baspl++ -t shell.b2m

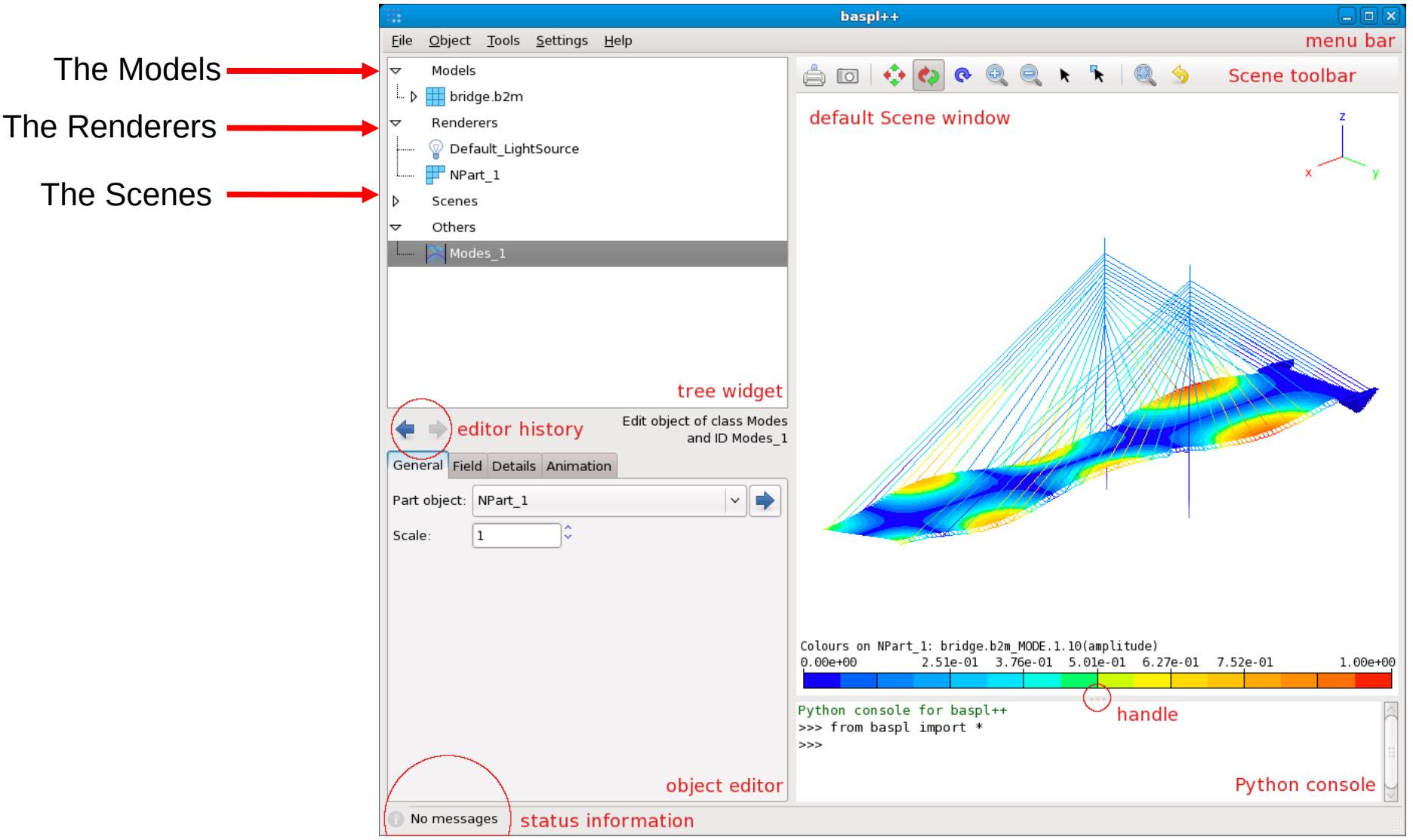
  which launches the viewer with the baspl++ interpreter (Python console) in the launching shell and

  baspl++ shell.b2m

  which launches the viewer with the interpreter (Python console) in the GUI (see next slide).

The baspl++ Viewer: Objects

baspl++ top-level window (object):

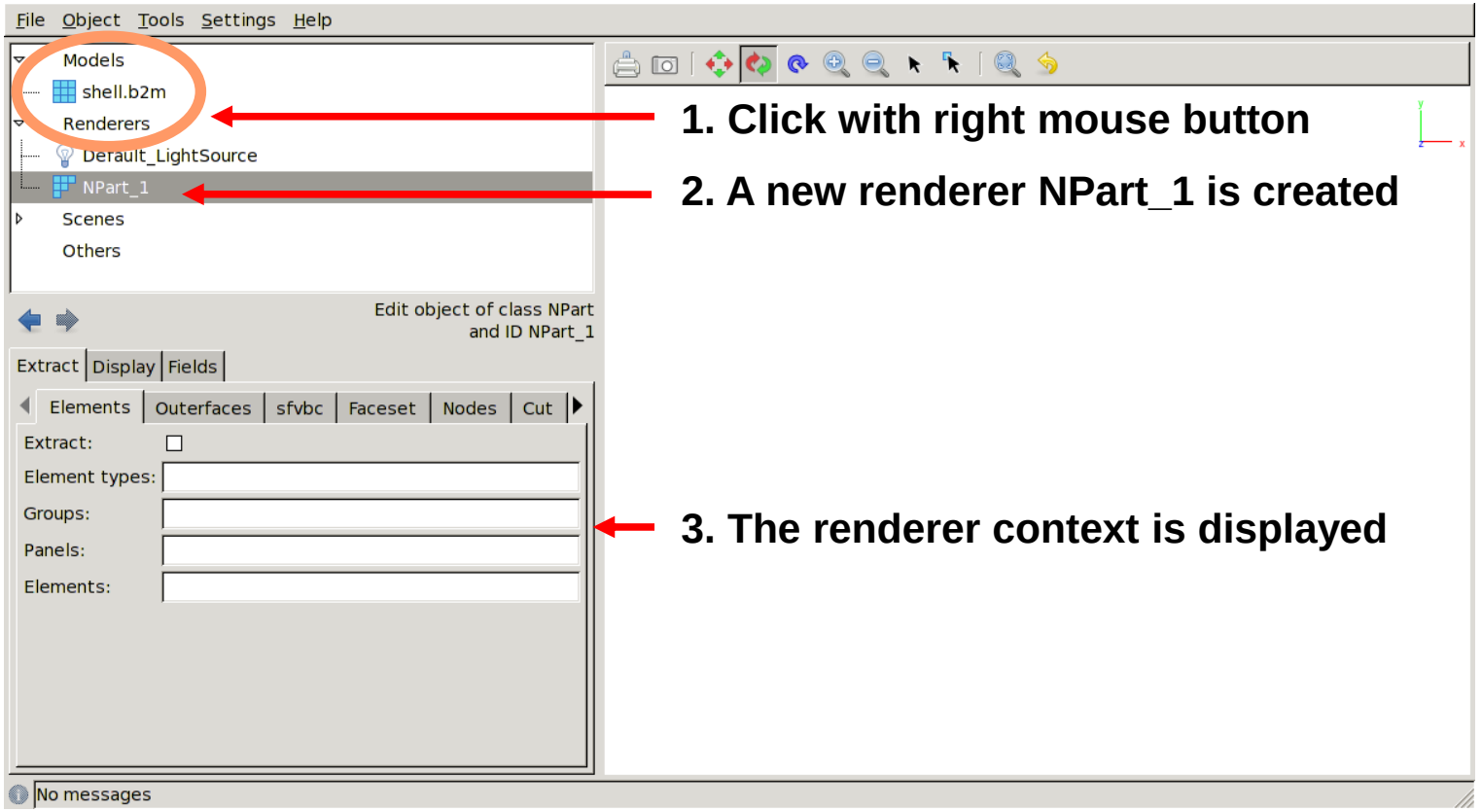


The baspl++ Viewer: Objects

- baspl++ main objects (classes):
- Some objects for viewing meshes and solutions:
  - **Model** (Models): Contains a B2000++ model database with mesh.
  - **Field**: Contains a solution field.
  - **NPart** (Renderers): Extracts or cuts – for rendering - elements, nodes, interpolates solution fields.
  - **Tracer**: Computes 3D particle traces from Model object and velocity solution field(s) and renders them.
- Objects for XY (graph) plotting
  - **Curve**: A container for x/y-values.
  - **Graph**: Contains a set of Curve objects and renders them.
- Applets and tools.
- Plus others (see baspl++ user manual).

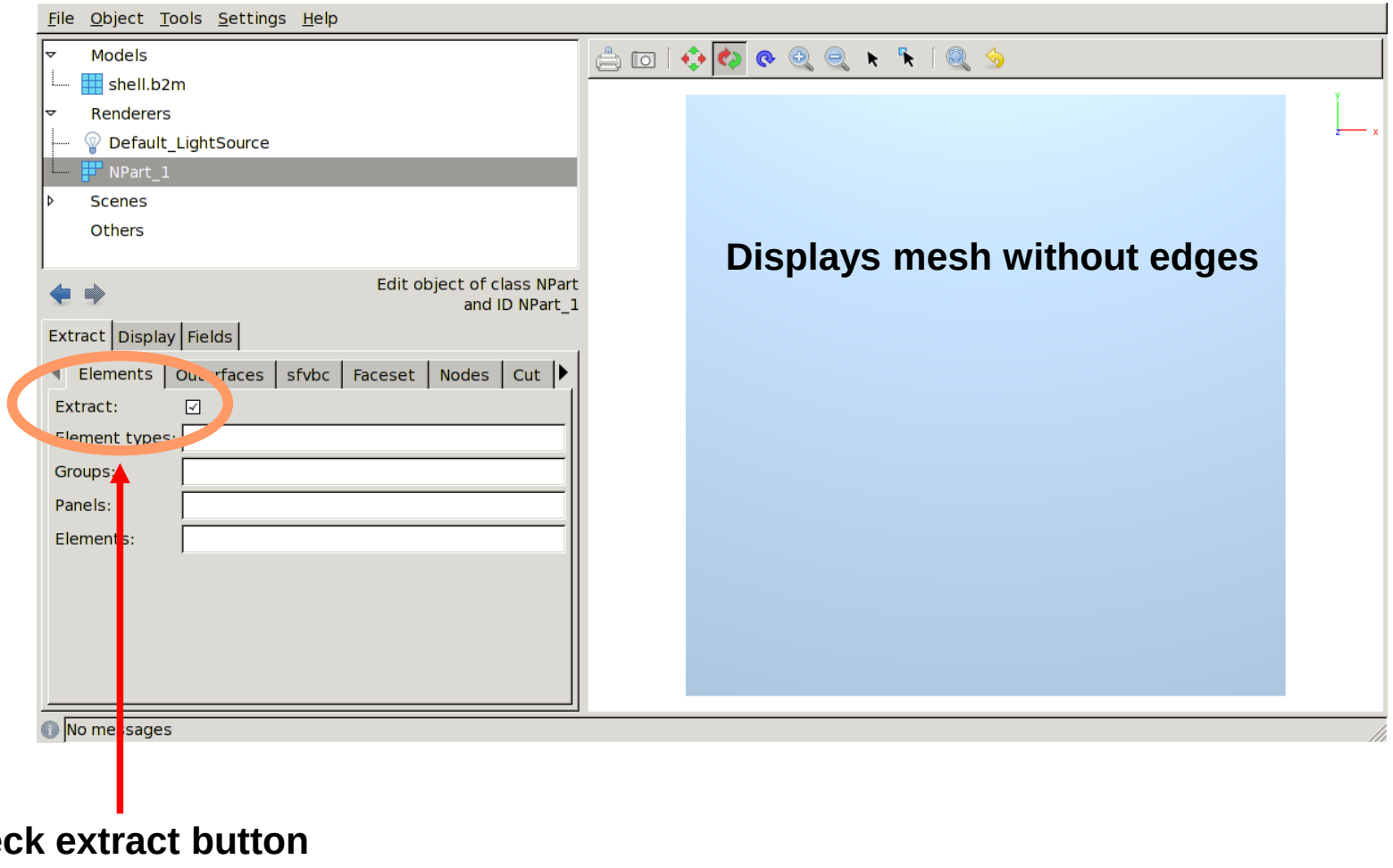
The baspl++ Viewer: View mesh

- To view a mesh and solutions
  - Open a B2000++ model database or pass the database name with the command line.
  - With the *right mouse button*, select *Create empty NPart* from the Model *shell.b2m*:



The baspl++ Viewer: View mesh

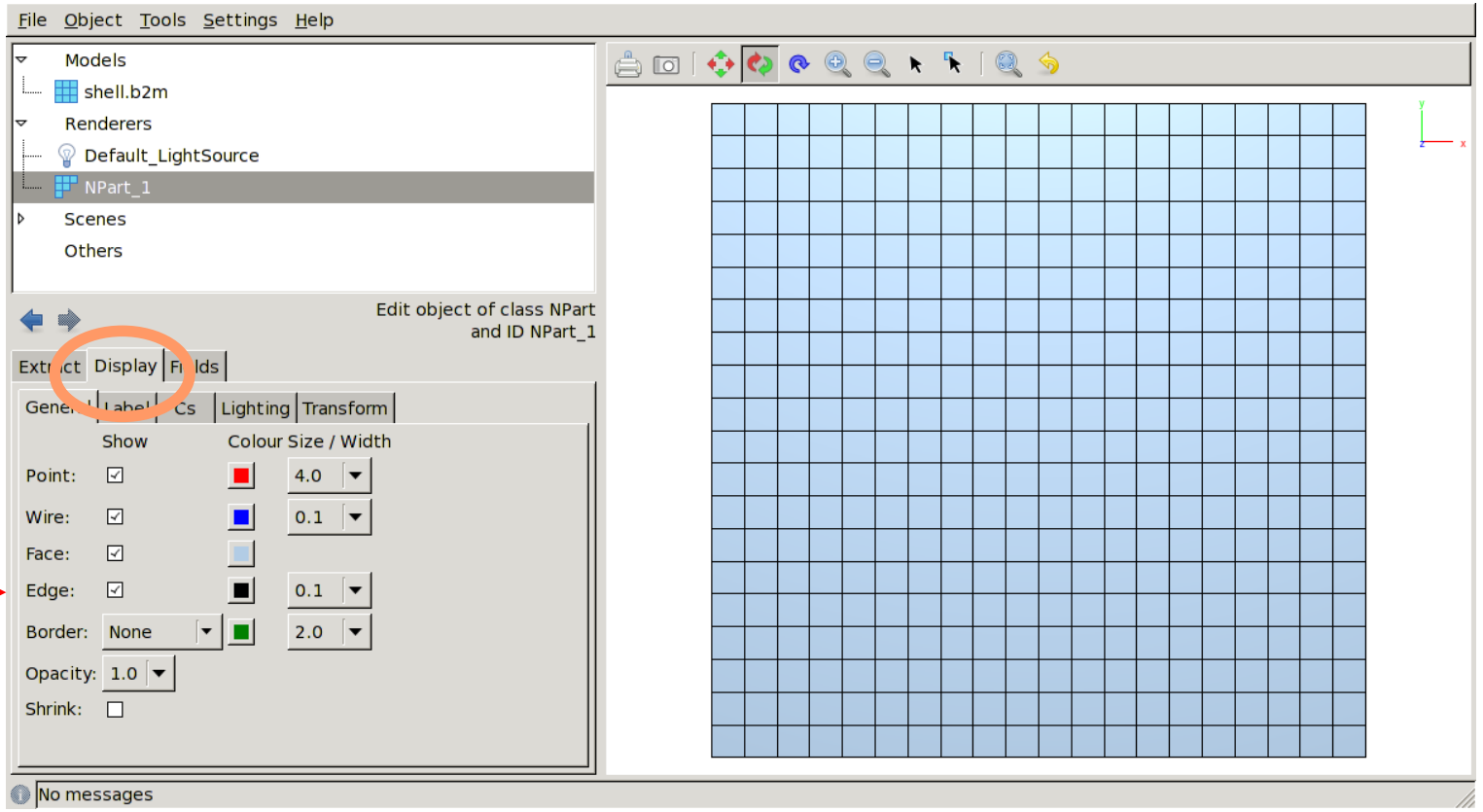
- Extract all elements:



Post-Processing
The baspl++ Viewer: View mesh

- Select *Display* menu and check *Edge* box: View edges

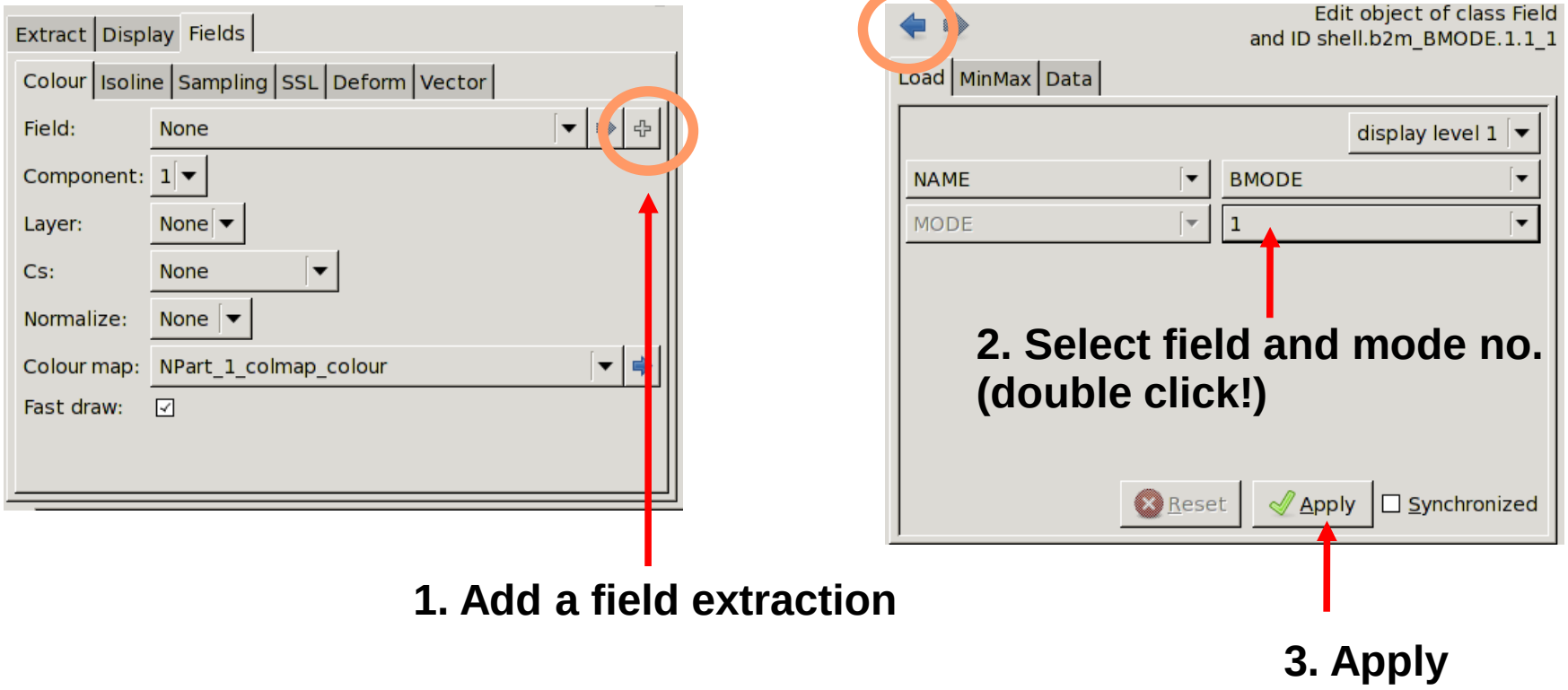
1. Check edge box



2. Displays mesh with edges

The baspl++ Viewer: View solution

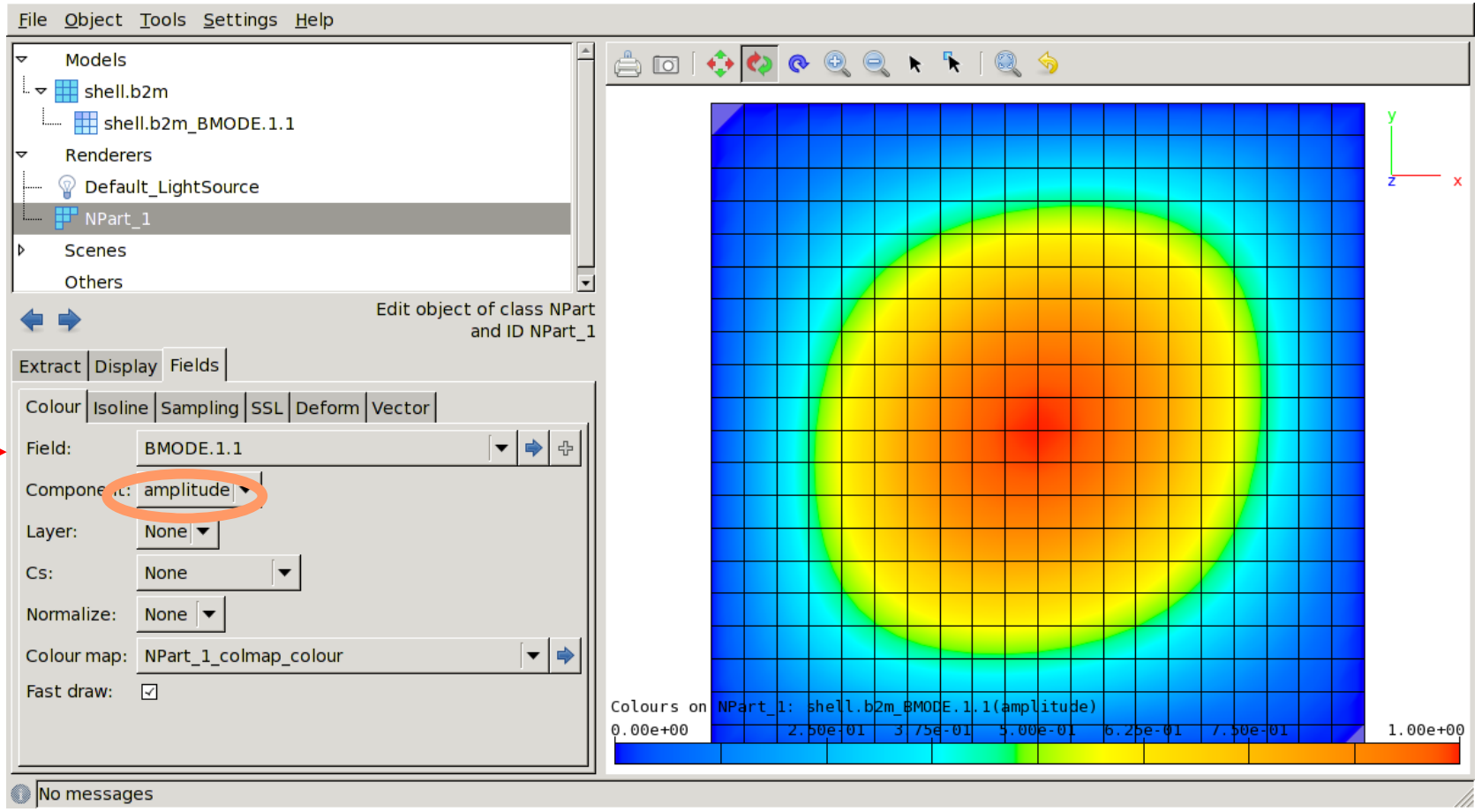
- To add i.e. amplitude of buckling mode, start by extracting the mode in the *Field* menu:



The baspl++ Viewer: View solution

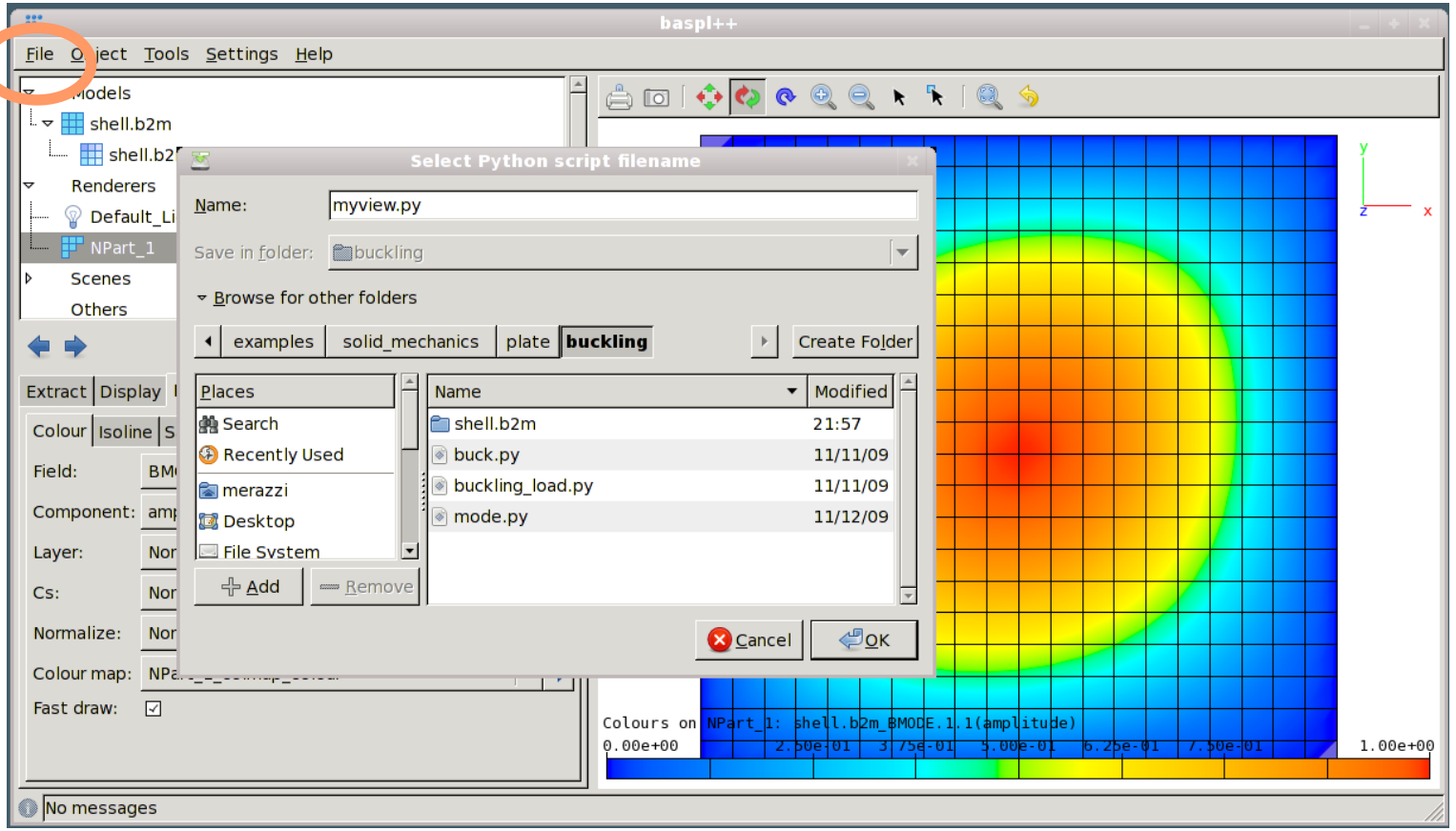
- To view amplitude of buckling mode, select extracted field (*BMODE.1.1*) and check *amplitude* in *Component* menu:

Select field



The baspl++ Viewer: Save Script

- To save the script which generates the current view, select *Create script* from *File* menu and type a file name:



- Restart with file name and restore view:

```
baspl++ myview.py
```

The baspl++ Viewer: Scripting

Python-scripting example: Extract the buckling analysis eigenvalue and transform to critical load (see file **`buckling_load.py`** in the buckling examples directory). The script

- Creates a baspl++ **Modes** objects, which computes, for the particular case of a *displacement loading*
  - the total reaction force F_x due to the *forced displacement.*
  - the total critical load F_{crit}

    $$F_{crit}=F_x * Eigenvalue$$
- Computes the critical load per unit length of the plate N_{crit}

  $$N_{crit}=F_{crit} / L$$

  where L is the plate edge length. Python code:

```
o = Modes()    # Ceate Modes object
o.part = p     # Add current part to Model objects
Fcrit = o.details[0].critical_force # Get critial force
s.label = "Fcrit=Fx*Eigenvalue [N]=%g, \
Ncrit=Fcrit/L [N/mm]=%g" % (Fcrit, Fcrit/200.)
```